Capitolo 7

Introduzione alla programmazione di applet

di *Rick Darnell*

IN QUESTO CAPITOLO

Nonostante Java sia un linguaggio di programmazione a scopo generale adatto a numerosissimi compiti, ciò per cui la gente lo utilizza maggiormente è la programmazione di applet. Un *applet* è un programma di Java che può essere eseguito solo in un browser Web compatibile, quale ad esempio Netscape Navigator o Microsoft Internet Explorer.

Una delle prime esperienze con Java fu un applet dimostrativo rilasciato nel 1995 con il primo Java Development Kit, che mostrava delle teste che ruotavano, un personaggio di un cartone animato che faceva la ruota e così via. Oggi gli applet vengono utilizzati per compiti molto diversi dai soli obiettivi dimostrativi, ad esempio i seguenti.

- ✔ Titoli sportivi e informazioni simili a quelli delle telescriventi.
- ✔ Grafica animata.
- ✔ Videogiochi.
- ✔ Esami per studenti.
- ✔ Mappe di immagini che rispondono al movimento del mouse.

- Visualizzazioni di testo avanzate.
- Rapporti di database.

Ad esempio, l'applet Instant Ballpark della Instant Sports (`http://www.instantsports.com/ballpark.html`) utilizza dati in tempo reale di partite di baseball e aggiorna la visualizzazione grafica in base a ciò che avviene nella partita. I giocatori corrono alle basi, la palla si sposta nel punto in cui è stata colpita, mentre per i falli, per il rumore del pubblico e per altro vengono utilizzati effetti sonori. Il programma, così unico da ottenere un brevetto negli Stati Uniti, si rifà alla tradizione dei vecchi tempi del baseball di presentare tutte le azioni per giochi di strada muovendo figure metalliche a lato di un edificio. Oltre che per la cronaca delle partite dal vivo, Instant Ballpark può essere utilizzato per rivedere le azioni delle ultime partite.

Con l'HTML e con un linguaggio di programmazione di gateway come Perl, una pagina Web permette di aggiornare il testo della cronaca di una partita mentre questa viene giocata. Oltre al testo, Instant Ballpark offre una rappresentazione visiva di un gioco dal vivo e può rispondere immediatamente all'input dell'utente. Questa natura interattiva di Java viene utilizzata per fornire informazioni agli utenti Web in modo più interessante, per questo molti fornitori di siti offrono applet Java.

Con l'incoraggiamento dato dalla Microsoft e dalla Apple al software a finestre negli ultimi cinque anni, gli utenti si aspettano oramai che il software utilizzi gli elementi grafici delle finestre, quali i pulsanti e le barre di scorrimento. In Java, le finestre e le altre funzioni dell'interfaccia utente grafica sono gestite dall'Abstract Windowing Toolkit (AWT).

L'AWT è uno dei package più utili inclusi nell'API di Java, ma è anche uno dei più difficili da apprendere. È una serie di classi utilizzata per creare interfacce utente grafiche per applicazioni e applet di Java e permette ai programmatori di creare in modo semplice i seguenti elementi per i programmi:

- finestre e finestre di dialogo;
- menu a tendina;
- pulsanti, etichette, caselle di controllo e altri componenti semplici dell'interfaccia utente;
- aree di testo, barre scorrevoli e altri componenti più sofisticati dell'interfaccia utente;
- gestori di layout per controllare il posizionamento dei componenti dell'interfaccia utente.

L'AWT include inoltre classi che permettono di gestire la grafica, i caratteri e i colori e una classe `Event` per consentire ai programmi di rispondere agli eventi del mouse e all'input della tastiera.

Inizialmente, il Java Developer's Kit (JDK) e un editor di testo erano l'unica possibilità per i programmatori di Java di sviluppare applet e applicazioni. Oggi vi sono numerosi strumenti per sviluppare programmi con Java, inclusi SunSoft Java WorkShop, Symantec Café e Visual Café e Rogue Wave JFactory. Java WorkShop, Café e Visual Café includono stru-

menti per lo sviluppo con il mouse di finestre, finestre di dialogo e altri elementi comuni ai sistemi a finestre. JFactory permette di creare interfacce e può essere utilizzato con diversi ambienti di programmazione di Java.

Sicurezza

Gli applet di Java sono programmi che vengono eseguiti sul computer di un utente Web. Come si sa, qualsiasi cosa che può eseguire codice costituisce un rischio potenziale per la sicurezza, a causa dei danni che possono verificarsi. I virus possono danneggiare il file system di un computer e riprodursi in altri dischi, i cavalli di Troia possono mascherarsi come programmi utili mentre causano danni, e altri programmi possono essere scritti apposta per richiamare subdolamente informazioni. Perfino le applicazioni Microsoft, come Word e Access, presentano rischi per la sicurezza a causa di Visual Basic for Applications, un linguaggio di programmazione eseguibile che può essere utilizzato con i documenti delle applicazioni.

La sicurezza è una delle preoccupazioni principali degli sviluppatori di Java, che hanno implementato dei meccanismi di protezione a diversi livelli. Alcuni di questi influiscono sul linguaggio in sé: rimozione di puntatori, verifica del bytecode e accesso ristretto a file remoti e locali. Ciò significa che alcune funzionalità di Java sono bloccate per gli applet, a causa dei problemi per la sicurezza.

- Gli applet non possono leggere o scrivere file sul disco dell'utente Web. Se occorre salvare delle informazioni sul disco durante l'esecuzione di un applet (ad esempio nel caso di un videogioco che salva i 10 migliori punteggi), la memorizzazione deve essere effettuata sul disco da cui proviene la pagina Web.
- Gli applet non possono effettuare connessioni di rete a computer diversi da quello da cui proviene la pagina Web, se non per indirizzare il browser in una nuova locazione.
- Le finestre aperte dagli applet sono identificate chiaramente come finestre di Java. Sul bordo di queste finestre viene visualizzata un'icona a forma di coppa e del testo, ad esempio `Untrusted Applet Window`, per evitare che una finestra aperta da Java pretenda di essere qualcos'altro, ad esempio una finestra di dialogo di Windows che richiede il nome e la password dell'utente.
- Gli applet non possono utilizzare librerie dinamiche o condivise di altri linguaggi di programmazione. Nonostante nelle applicazioni di Java sia possibile utilizzare programmi scritti in altri linguaggi, quali Visual C++, negli applet non si può utilizzare questa funzionalità, in quanto non vi è modo di verificare adeguatamente la sicurezza del codice diverso da quello di Java.
- Gli applet non possono eseguire programmi nel sistema dell'utente Web, inclusi i plug-in dei browser, i controlli ActiveX o altri elementi correlati ai browser.

Queste restrizioni fanno sì che gli applet di Java siano più limitati rispetto alle applicazioni autonome. Ciò rappresenta una compensazione per la sicurezza che deve essere messa in atto affinché il linguaggio possa essere eseguito a distanza sui computer degli utenti. Nel Capitolo 29 vengono fornite ulteriori informazioni sulla sicurezza in Java.

Fondamenti della programmazione di applet

Ora che si sa che cosa sono gli applet, è il momento di utilizzare alcuni potenti strumenti e capire che cosa è necessario fare per creare un applet. Di seguito viene prima presentata una breve introduzione ad alcuni degli elementi fondamentali della programmazione degli applet.

Ogni applet inizia con una definizione di classe come la seguente:

```
public class ImparaLatino extends java.applet.Applet {
    // da fare
}
```

In questo esempio, `ImparaLatino` è il nome della classe dell'applet. Un applet deve essere *dichiarato come classe pubblica. Tutti gli applet estendono la classe* `java.applet.Applet`, chiamata anche classe `Applet`. Un altro modo per scrivere la definizione di classe è quello di utilizzare `import` per caricare la classe `Applet`, semplificando la dichiarazione:

```
import java.applet.Applet
public class ImparaLatino extends Applet {
    // da fare
}
```

`Applet` estende la classe `java.awt.Panel`, che è un contenitore di Java. Questa evoluzione permette agli applet di mantenere un posto fisico nella pagina Web nella finestra del browser. Uulteriori informazioni si trovano più avanti in questo capitolo.

Le superclassi di `Applet` forniscono a tutti gli applet una struttura su cui è possibile costruire gli elementi dell'interfaccia utente e gli eventi del mouse, e anche una struttura per l'applet che deve essere utilizzata quando viene sviluppato il programma.

Metodi degli applet

La struttura di un'applet è determinata da quattro eventi che hanno luogo durante la sua vita. Quando viene raggiunto ogni stadio, viene richiamato automaticamente un metodo. Esistono versioni predefinite per ognuno dei metodi, se si decide di non ridefinirli.

- `init()`: è il metodo di inizializzazione utilizzato per impostare lo stadio per l'attività dell'applet; questo metodo di norma include il caricamento della grafica, l'inizializzazione di variabili e la creazione di oggetti.
- `start()`: di seguito viene iniziata l'esecuzione dell'applet. Questo è il fulcro dell'applet, dove avviene ogni cosa. Il metodo `start()` è il corpo dell'applet ed è utilizzato anche per riavviare l'applet dopo che è stato arrestato.

Applet e applicazioni

Questo capitolo si concentra sugli applet, ma è importante chiarire la distinzione tra i due tipi di programmi di Java. Gli *applet* sono programmi inclusi in pagine Web che per essere eseguiti richiedono l'utilizzo di un browser Web. Le *applicazioni* sono tutto il resto: programmi di carattere generale eseguiti per mezzo dell'interprete di Java con il nome del programma come argomento. Ad esempio, per eseguire il programma di Java `LeggiNews.class`, alla riga di comando si immette:

```
java LeggiNews
```

Per le applicazioni non valgono molte delle restrizioni degli applet, nonostante siano comunque implementate funzionalità quali la verifica del bytecode.

Molti programmatori inizialmente sono confusi dall'assenza di un metodo `main()` per iniziare un applet. È importante capire che il browser fornisce tutto il supporto e la responsabilità necessari per l'esecuzione dell'applet. In altre parole, il browser è il metodo `main()` e l'applet è una subroutine che viene eseguita nel suo contesto.

Per ulteriori informazioni sulle tecniche per lavorare con i programmi di Java, si rimanda alla Parte 4 del libro.

- `stop()`: arrestando un'applet se ne interrompe l'esecuzione, ma si lasciano intatte le risorse, in modo da poterlo riavviare. Si dovrebbe sempre arrestare un applet prima di distruggerlo; è possibile utilizzare il metodo `stop()` anche per arrestare l'esecuzione di un applet quando è necessaria una pausa nel flusso.
- `destroy()`: quando si distrugge un applet, tutte le sue risorse, quali la memoria, il tempo del processore e lo spazio su disco utilizzato come file di scambio, vengono liberate e restituite al sistema. Questo metodo è l'ultima cosa che può accadere quando l'utente lascia la pagina che contiene l'applet.

Sebbene non siano direttamente necessari per l'esecuzione di un programma, vi sono due altri metodi importanti, `paint()` e `repaint()`, che vengono utilizzati solo per la visualizzazione nella finestra dell'applet e che sono eseguiti automaticamente in determinati momenti, ad esempio quando la finestra dell'applet viene nuovamente visualizzata dopo che è stata nascosta o quando ne vengono modificate le dimensioni. Gli applet richiamano direttamente `repaint()` per aggiornare la finestra ogni volta che è necessario.

Di tutti i metodi, `repaint()` è l'unico che necessita di un parametro. Questo parametro è un'istanza della classe `Graphics`, come indicato nella seguente definizione di metodo:

```
public void paint(Graphics g) {
   g.drawString("Un momento, per favore", 5, 50);
}
```

L'oggetto Graphics utilizzato come parametro rappresenta la finestra dell'applet. La riga g.drawString() utilizza questo oggetto Graphics per indicare dove deve essere disegnata una stringa. Ogni volta che viene richiamato il metodo repaint(), la finestra dell'applet viene aggiornata con la stringa: "Un momento, per favore" disegnata nella posizione determinata da x e y (5, 50).

Ognuno di questi metodi init(), destroy(), start(), stop() e paint(), viene ereditato dagli applet. Tuttavia, tutti i metodi degli applet, in base alle impostazioni predefinite, sono vuoti; se si suppone che in un applet avverrà qualcosa di specifico, è necessario ridefinire alcuni o tutti i metodi. Nonostante non sia necessario ridefinire questi metodi, come regola generale si fornisce sempre un metodo start() personalizzato.

Visualizzazione di applet

Gli applet vengono visualizzati come parte di una pagina Web utilizzando il tag HTML <APPLET>. Per eseguire un applet, è necessario un browser Web o un altro software che abbia la funzione di un browser, ad esempio il programma di visualizzazione di applet fornito con il Java Development Kit da JavaSoft.

Il browser agisce da sistema operativo per l'applet: non è possibile eseguire un applet come programma autonomo nello stesso modo in cui si esegue un file eseguibile. Due browser importanti, Netscape Navigator (versione 2.02 e successive) e Microsoft Internet Explorer (versione 3.0 e successive), supportano entrambi gli applet di Java. Vi è una terza possibilità, HotJava della Sun, per gestire gli applet, ma questo browser non è molto diffuso.

Questi programmi caricano gli applet da una pagina Web e li eseguono a distanza nel computer dell'utente Web. Ciò crea problemi relativi alla sicurezza, che devono essere gestiti dal linguaggio Java stesso e dai browser compatibili con Java.

Il tag <APPLET>

L'esecuzione di applet di Java in una pagina Web richiede l'utilizzo di due tag HTML speciali: <APPLET> e <PARAM>. Questi tag vengono inclusi in una pagina Web assieme al resto del codice HTML. Inserire un applet di Java non è diverso dall'inserimento di un'immagine, come indicano le seguenti righe:

```
<APPLET CODE="OraMostra.class" CODEBASE="progdir" WIDTH=376 HEIGHT=104>
<PARAM NAME="speed" value="100">
<PARAM NAME="blink" value="5">
<PARAM NAME="text" value="GRATIS UN BUONO PER UNA PIZZA!">
<PARAM NAME="fontsize" value="21">
<PARAM NAME="pattern" value="random">
<H5>Questo applet richiede un browser compatibile con Java!</H5>
</APPLET>
```

Quando viene incluso in una pagina Web, questo codice HTML fa in modo che in un browser compatibile con Java avvenga quanto segue.

Perché passare i numeri come stringhe?

Tutti i parametri passati da una pagina HTML a un applet vengono passati come stringhe, indipendentemente da che cosa sia il parametro o da come sia formattato nella pagina. Qualsiasi conversione in altri tipi, ad esempio `integer`, `boolean` o `date`, deve avvenire all'interno dell'applet.

Ad esempio, `<PARAM NAME="speed" value=100>` e `<PARAM NAME="speed" value="100">` passano entrambi la stringa `"100"` all'applet. Questa restrizione evita al browser e all'applet del lavoro inutile, in quanto nessuno dei due deve indovinare quale tipo di valore viene inviato. Tutto viene convertito in stringa e il programmatore può occuparsi della conversione del valore in un nuovo tipo, dopo che questo è entrato nell'applet.

1. Un applet chiamato `OraMostra.class` viene caricato da una directory chiamata `progdir`. L'attributo `CODE` specifica il nome di file dell'applet, mentre l'attributo opzionale `CODEBASE` fa riferimento a una directory in cui si trova l'applet.
   Se l'attributo `CODEBASE` viene utilizzato senza la barra (/), viene indicato il percorso dalla directory della pagina Web alla directory che contiene il file di classe dell'applet. Ad esempio `CODEBASE="usr"` indica che l'applet è in una directory chiamata `usr`, che è una sottodirectory della directory della pagina Web. `CODEBASE` può anche specificare un percorso completo nel sito Web separato dalla locazione corrente della pagina.
2. L'applet viene impostato con una larghezza di 376 pixel e un'altezza di 104 pixel utilizzando gli attributi `WIDTH` e `HEIGHT`, che funzionano nello stesso modo sia con `<APPLET>` che con `<IMG>`.
   L'attributo `ALIGN` utilizzato con le immagini può essere utilizzato anche con `<APPLET>` e determina il modo in cui l'applet viene posizionato in relazione alle altre parti della pagina Web; i valori possibili sono `TOP`, `MIDDLE` o `BOTTOM`.
3. All'applet vengono inviati un parametro chiamato `speed` con un valore di `100` e quattro altri parametri: `blink`, `text`, `fontsize` e `pattern`. I parametri sono opzionali ed è possibile includerne quanti se ne desiderano. L'attributo `NAME` indica il nome da attribuire a un parametro, mentre l'attributo `VALUE` indica il valore da associare.
4. La riga `<H5>Questo applet richiede un browser compatibile con Java!</H5>` viene ignorata a meno che la pagina venga caricata da un browser non compatibile con Java. Un browser non compatibile non riconosce i due tag `<APPLET>` e ignora tutte le righe finché raggiunge il tag dei titoli `<H5>`. Poiché sa come gestire questo tag, il browser visualizza il testo appropriato.

È possibile inserire qualsiasi codice HTML prima del tag di chiusura </APPLET>. Molti progettisti di pagine includono un'immagine o un elemento simile che occupa lo stesso spazio dell'applet. In questo modo si ha un layout di pagina uniforme, indipendentemente dal browser utilizzato.

Se l'applet utilizza file di classe che non fanno parte dell'API standard di Java, questi file devono trovarsi nella stessa directory del file di classe dell'applet.

Inserimento di applet nel Web

Rendere disponibile un applet nel Web dopo averlo creato e inserito in una pagina HTML con i tag `<APPLET></APPLET>` è semplice: è sufficiente inserire tutti i file `.class` necessari per l'applet nel sito Web, assicurandosi che siano nella directory specificata dall'attributo `CODEBASE`, se è stato utilizzato. Se l'attributo `CODEBASE` non è stato utilizzato, i file `.class` devono essere inseriti nella stessa directory della pagina Web che include l'applet.

Non serve nient'altro. A differenza della programmazione CGI, che necessita di un accesso speciale al computer in cui si trovano le pagine Web, gli applet di Java possono essere aggiunti da chiunque sia in grado di inserire pagine in un sito Web.

Altre informazioni sull'utilizzo dei parametri

Come si è visto precedentemente, i parametri vengono inviati a un applet per mezzo del tag `<PARAM>` e dei suoi attributi `NAME` e `VALUE`. Ad esempio, `<PARAM NAME="blink" VALUE="100">` invia all'applet un parametro di nome `blink` con un valore di `100`.

A differenza di tutto il resto riguardante Java, per il nome del parametro non vi è differenza tra caratteri maiuscoli e minuscoli: è possibile inviare all'applet indistintamente una `rosa`, una `Rosa`, o una `ROSA`.

I parametri vengono passati all'applet dopo che questo è stato caricato; tutti i parametri vengono inviati come stringhe, che siano racchiusi tra virgolette o meno, e vengono convertiti in altri tipi di dati all'interno dell'applet. Affinché un applet possa utilizzare un parametro, lo deve richiamare utilizzando il metodo `getParameter()`, che di norma viene impiegato all'interno della parte `init()` dell'esecuzione dell'applet.

Ad esempio, la seguente riga di un applet di Java viene utilizzata per memorizzare il parametro `blink` in una variabile chiamata `blinkValue`:

```
String blinkValue = getParameter("blink");
```

Per richiamare il valore e convertirlo in un intero, si utilizza il seguente codice:

```
int blinkValue = -1;
try { blinkValue = Integer.parseInt(getParameter("blink")); }
catch (NumberFormatException e) { }
```

Questo esempio utilizza il metodo `parseInt()` della classe `java.lang.Integer` per convertire un valore `String` in `int`. Il blocco `try` e `catch` viene utilizzato per rilevare gli errori se la stringa non può essere convertita in un numero.

Un esempio: l'applet ColorCycle

Nei prossimi capitoli vengono approfonditi i dettagli specifici della programmazione, inclusa la progettazione dell'interfaccia utente e la gestione degli eventi. Per il momento, si osservi il seguente esempio per avere un'idea più chiara di come sono costruiti gli applet.

`ColorCycle` è un semplice applet con un pulsante chiamato `Colore seguente`. Quando si fa clic su questo pulsante con il mouse, il colore dello sfondo dell'applet cambia. Questo programma mostra la struttura di base degli applet e un semplice esempio di gestione degli eventi. Il Listato 7.1 include il codice sorgente di questo applet, che si trova anche nel CD-ROM allegato al libro.

Listato 7.1 *Il codice sorgente di ColorCycle.java.*

```
 1: import java.awt.*;
 2:
 3: public class ColorCycle extends java.applet.Applet {
 4:     float hue = (float).5;
 5:     float saturation = (float)1;
 6:     float brightness = (float)0;
 7:     Button b;
 8:
 9:     public void init() {
10:         b = new Button("Colore seguente");
11:         add(b);
12:     }
13:
14:     public void start() {
15:         setBackground(Color.black);
16:         repaint();
17:     }
18:
19:     public boolean action(Event evt, Object o) {
20:         if (brightness < 1)
21:             brightness += .25;
22:         else
23:             brightness = 0;
24:         Color c = new Color(Color.HSBtoRGB(hue, saturation, brightness));
25:         setBackground(c);
26:         repaint();
27:         return true;
28:     }
29: }
```

Non ci si deve preoccupare se alcuni aspetti di questo programma non sono ancora familiari. Diverse parti di questo applet vengono discusse più avanti in questo capitolo, inclusa la creazione di componenti dell'interfaccia utente quali i pulsanti e il metodo `action()`.

- Riga 1: l'applet importa diverse classi utilizzando il carattere jolly con `java.awt.*`; `awt` è l'abbreviazione di Abstract Windowing Toolkit ed è una serie di classi che gestisce la maggior parte degli aspetti visivi e interattivi della programmazione in Java: la grafica, i caratteri e l'input per mezzo della tastiera e del mouse.

- **Righe da 4 a 6:** vengono create tre variabili istanza per memorizzare i valori HSB del colore visualizzato. HSB (Hue, Saturation e Brightness – tonalità, saturazione e luminosità) è un metodo per descrivere un colore con tre valori numerici da `0` a `1`.
- **Riga 7:** viene creato un oggetto `Button`.
- **Righe da 9 a 12:** quando l'applet viene eseguito per la prima volta, viene richiamato automaticamente il metodo `init()`, che permette l'istanziazione dell'oggetto `Button b` con l'etichetta `Colore seguente`.
- **Righe da 14 a 17:** il metodo `start()` imposta il colore di sfondo dell'applet sul nero, utilizzando una costante della classe `Color` (`Color.black`). Inoltre viene richiamato il metodo `repaint()` per ridisegnare la finestra dell'applet, poiché è cambiato qualcosa (il colore di sfondo) e si desidera visualizzare questo cambiamento.
- **Riga 19:** il metodo `action()` viene richiamato ogni volta che un componente dell'interfaccia utente genera un evento di azione. In questo applet, quando si fa clic sul pulsante si verifica un evento. Per ulteriori informazioni sulla gestione degli eventi, si rimanda ai Capitoli 6 e 21.
- **Righe da 20 a 23:** il valore di `brightness` viene modificato in modo che il colore di sfondo vari da nero a diverse tonalità di azzurro.
- **Righe da 24 a 26:** viene creato un oggetto `Color` per memorizzare il valore del colore di sfondo, specificato sulla base dei valori delle variabili `hue`, `saturation` e `brightness`. Il nuovo colore di sfondo viene visualizzato richiamando nuovamente `repaint()`.
- **Riga 27:** alla fine del metodo `action()` viene restituito il valore booleano `true`, che indica che è stato gestito l'evento di azione generato facendo clic sul pulsante.

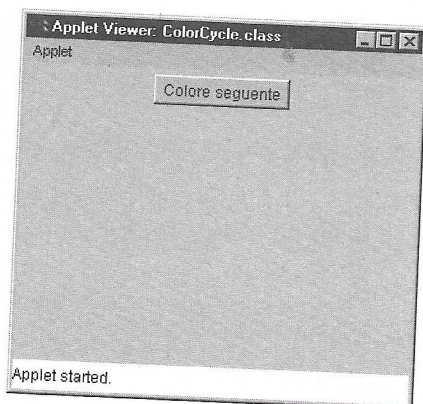
Si noti che in questo applet non sono stati definiti metodi `stop()` o `destroy()`, in quanto non è stato necessario ridefinire le attività di questi due metodi. Quando l'utente passa a una pagina Web diversa o rimuove l'applet dal visualizzatore, l'applet termina e vengono recuperate le risorse.

Progettazione della pagina HTML per ColorCycle

Dopo aver scritto e compilato l'applet `ColorCycle` utilizzando un software di sviluppo, è possibile inserirlo in una pagina Web utilizzando i tag HTML `<APPLET>`, `</APPLET>` e `<PARAM>` descritti precedentemente in questo capitolo.

Il Listato 7.2 mostra il testo completo di una pagina HTML che carica l'applet `ColorCycle.class` *(questo codice si trova anche nel CD-ROM allegato al libro). Poiché con il tag* `<APPLET>` *non viene utilizzato l'attributo* `CODEBASE`*, il file* `.class` *deve trovarsi nella stessa directory della pagina Web che contiene l'applet.*

Nonostante l'applet perda qualcosa nella stampa in bianco e nero, la Figura 7.1 mostra come appare l'applet `ColorCycle` quando lo si visualizza con il visualizzatore di applet fornito con il Java Development Kit dalla Sun.

Figura 7.1
L'applet ColorCycle è costituito da una cornice quadrata il cui sfondo cambia quando si fa clic sul pulsante.

Listato 7.2 *Il codice sorgente di ColorCycle.html.*

```
1: <html>
2: <body>
3: <applet code=ColorCycle.class height=250 width=250>
4: </applet>
5: </body>
6: </html>
```

Trasformazione dell'applet ColorCycle in un'applicazione

Per evidenziare le differenze tra applet e applicazioni, l'applet `ColorCycle` *viene ora trasformato in un'applicazione che può essere eseguita senza l'utilizzo di un browser (si veda il Listato 7.3 e il codice nel CD-ROM). È opportuno ricordare che l'applicazione deve fornire la struttura di base normalmente fornita per l'applet dal browser e che deve anche richiamare a rotazione tutti i metodi dell'applet.*

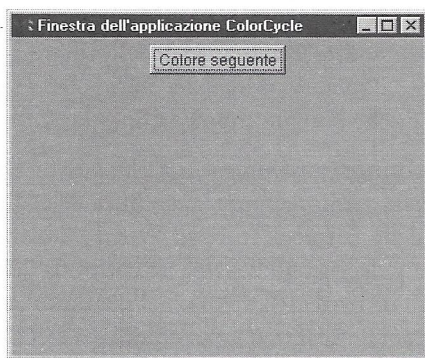
Listato 7.3 *Il codice sorgente dell'applicazione ColorCycleApplication.java.*

```
1:  import java.applet.Applet;
2:  import java.awt.*;
3:
4:  public class ColorCycleApplication extends Applet {
5:     float hue = (float).5;
6:     float saturation = (float)1;
7:     float brightness = (float)0;
8:     Panel p;
9:     Button b;
10:
11:   public static void main(String args[]) {
12:    ColorCycleFrame app = new ColorCycleFrame("Finestra dell'applicazione
                                              ColorCycle");
13:       app.resize(200,200);
14:       app.show();
15:       app.applet.start();
```

```
16:     }
17:
18:     public void init() {
19:          b = new Button("Colore seguente");
20:          add(b);
21:     }
22:
23:     public void start() {
24:          setBackground(Color.black);
25:          repaint();
26:     }
27:
28:     public boolean action(Event event, Object obj) {
29:          if (brightness < 1)
30:              brightness += .25;
31:          else
32:              brightness = 0;
33:          Color c = new Color(Color.HSBtoRGB(hue, saturation, brightness));
34:          setBackground(c);
35:          repaint();
36:          return true;
37:     }
38: }
39:
40: class ColorCycleFrame extends Frame {
41:     ColorCycleApplication applet;
42:
43:     public ColorCycleFrame(String frameName) {
44:          super(frameName);
45:          applet = new ColorCycleApplication();
46:          add("Center",applet);
47:          applet.init();
48:     }
49:
50:     public boolean handleEvent(Event event) {
51:          if(event.id == Event.WINDOW_DESTROY) {
52:               applet.stop();
53:               applet.destroy();
54:               System.exit(0); }
55:          return false;
56:     }
57: }
```

All'applet `ColorCycle` vengono aggiunti un'ulteriore classe e un ulteriore metodo, in modo che la struttura normalmente messa a disposizione dal browser venga ora fornita dall'applicazione stessa. In questo modo si permette l'esecuzione dello stesso codice sia come applet che come applicazione (si veda la Figura 7.2).

Nel Listato 7.3 l'applicazione inizia importando le classi necessarie per supportare entrambi i tipi di operazione (righe 1 e 2). Le classi importate includono la classe `Applet` per l'applet e la classe `awt.Frame` per l'applicazione.

Figura 7.2
L'applet ColorCycle trasformata in un'applicazione indipendente.

Poiché la classe `Frame` *fa parte dell'AWT, è più semplice importarla con le altre classi dell'AWT utilizzando l'istruzione* `import java.awt.*`*. Anziché scrivere un lungo elenco di istruzioni* `import` *con ogni singolo elemento necessario, quando in un programma vengono utilizzate due o tre sottoclassi si dovrebbe sfruttare questo tipo di approccio. In questo modo si ottiene un codice più pulito e meno possibilità di errori di digitazione quando il codice viene compilato.*

Il passaggio successivo consiste nell'assicurarsi che il codice possa essere eseguito in un browser: nella riga 4, la classe `ColorCycleApplication` estende la classe `Applet`. Successivamente viene inserito come primo metodo della classe il metodo `main()` richiesto dalle applicazioni, che crea e visualizza una cornice per contenere il programma (riga 11), la quale a sua volta inizia il corpo del programma. Se il programma viene eseguito come applet, `main()` non viene utilizzato e il browser inizia con il metodo `init()`; se la classe invece viene eseguita come applicazione, viene prima eseguito il metodo `main()`, che crea una cornice per contenere le funzioni dell'applet utilizzando la classe `ColorCycleFrame`. Dopo aver creato la cornice, l'applet viene centrato al suo interno.

Si ricordi che un browser di norma fornisce l'intera struttura necessaria a un applet e che è anche responsabile di richiamare i quattro metodi necessari: `init()`, `start()`, `stop()` e `destroy()`. Tuttavia, quando il programma viene eseguito come applicazione, è necessario sostituire questa struttura, perché il browser non è disponibile. Questi dettagli vengono gestiti principalmente dal metodo `ColorCycleApplication.main()` e dalla classe `ColorCycleFrame`. Si osservi più attentamente quest'ultima classe.

Quando un applet viene eseguito in un browser, quest'ultimo fornisce una cornice in cui risiede l'applet. Non essendo un browser, l'applicazione non è a conoscenza di questo requisito, pertanto la classe `ColorCycleFrame` fornisce la prima parte della struttura necessaria. Di seguito, il programma deve fornire un'istanza della classe dell'applicazione da utilizzare come applet (riga 41). Le righe da 33 a 46 creano la cornice fisica e vi aggiungono un'istanza di `ColorCycleApplication`. Per preparare l'applet per il funzionamento viene richiamato il primo metodo dell'applet, `init()`.

Dopo aver creato un'istanza della cornice e dell'applet, il controllo passa di nuovo al metodo `main()`. Le righe 13 e 14 impostano le dimensioni iniziali della finestra e la visualizzano sullo schermo. Viene quindi richiamato il metodo principale dell'applet, `start()`. A questo punto il programma viene eseguito finché l'utente chiude la finestra.

Qualsiasi azione riguardante la cornice causa un evento, che può essere lo spostamento del mouse all'interno o all'esterno della cornice oppure la modifica delle dimensioni, la riduzione a icona o l'ingrandimento della cornice. Tutti gli eventi vengono gestiti utilizzando `ColorCycleFrame.handleEvent()` nella riga 50. La riga 51 controlla l'evento per vedere se si tratta della chiusura della finestra da parte dell'utente, nel qual caso vengono richiamati gli ultimi due metodi dell'applet, `stop()` e `destroy()`, per chiudere l'esecuzione del programma e uscire dal sistema Java.

I metodi `stop()` *e* `destroy()` *vengono richiamati, ma non sono definiti esplicitamente nella classe* `ColorCycleApplication`. *Questo perché la classe* `Applet` *include le definizioni predefinite di questi due metodi. Se uno di essi non è incluso come parte di una classe definita dall'utente, il sistema di Java richiama automaticamente la versione predefinita in* `Applet`.

Aggiungendo a qualsiasi applet la classe della cornice e dell'ulteriore codice, è possibile eseguire l'applet su computer che non hanno browser compatibili, ma che hanno una macchina virtuale di Java.

L'AWT di Java

Dopo aver capito la struttura di base in cui si sviluppano gli applet, si può dare un'occhiata più da vicino all'Abstract Windowing Toolkit (AWT), utilizzato per creare l'interfaccia utente grafica con elementi standard delle finestre. Ognuno di questi elementi è rappresentato da un componente; vi sono componenti per i pulsanti su cui è possibile fare clic, componenti per i campi di testo in cui è possibile digitare e componenti per le barre scorrevoli che possono essere controllate. Vi sono inoltre componenti per elementi che non possono essere manipolati direttamente, ad esempio le etichette.

Per ulteriori informazioni sulle funzionalità e sulle classi dell'AWT, si rimanda alla Parte 4.

I contenitori: un posto per i componenti

Per utilizzare i componenti dell'AWT in un programma, è necessario inserirli in qualcosa che li contenga: dei pulsanti che si spostano liberamente nel computer non sono per niente utili. Un *contenitore* di Java è come una lavagna vuota che contiene i componenti dell'interfaccia. Tutti i contenitori in Java sono sottoclassi della classe `Container`.

Vi sono due tipi fondamentali di contenitori.

- ✔ La classe `Window`: questa classe crea finestre separate dal programma principale e ha due sottoclassi: `Frame` (finestre che hanno un bordo e una barra dei menu) e `Dialog` (una finestra speciale utilizzata nelle applicazioni per selezionare un file).

- La classe `Panel`: contenitore che rappresenta una sezione di una finestra esistente. La classe `Applet` è un contenitore che è una sottoclasse della classe `Panel`. È possibile inserire i componenti direttamente nel pannello di un applet oppure utilizzare altri oggetti `Panel` per suddividere l'applet in sezioni più piccole.

Un contenitore `Panel` non è visibile quando viene aggiunto a un applet. Il suo scopo è permettere l'organizzazione dei componenti mentre vengono disposti in una finestra.

Il Listato 7.4 mostra il codice sorgente di un applet alla cui superficie viene aggiunto un nuovo `Panel`. L'applet si trova anche nel CD-ROM allegato al libro. Questo codice non produce un output, a eccezione di una finestra vuota, tuttavia è utile come modello per i componenti che si imparerà ad aggiungere in questo capitolo.

Listato 7.4 *Il codice sorgente di ContainerAndPanel.java.*

```
1: import java.applet.Applet;
2: import java.awt.*;
3: public class ContainerAndPanel extends Applet {
4:     Panel p = new Panel();
5:
6:     public void init() {
7:         add(p);
8:     }
9: }
```

L'istruzione che inserisce l'istanza `p` del panello nella finestra dell'applet è `add(p)` (riga 7). Il metodo `add()` viene utilizzato ogni volta che si desidera aggiungere un qualsiasi tipo di componente in un contenitore.

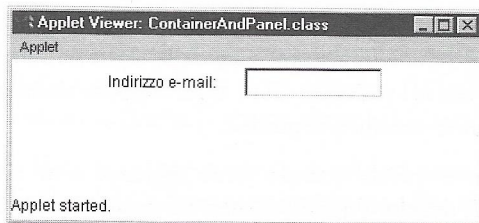
È possibile utilizzare questo codice sorgente per verificare ogni componente sostituendo `Panel p= new Panel()` con le dichiarazioni dei componenti desiderati e aggiornando la riga `add(p)`.

In base alle impostazioni predefinite, i componenti vengono aggiunti nei contenitori da sinistra a destra e dall'alto verso il basso. Se in una riga non vi è spazio sufficiente per un componente, questo viene inserito a sinistra nella riga successiva. Vi sono inoltre numerosi *gestori di layout* che permettono un maggior controllo del posizionamento dei componenti. L'organizzazione dei componenti per mezzo dei gestori di layout viene discussa più avanti in questo capitolo e nel Capitolo 14.

È arrivato il momento di analizzare i diversi componenti che si possono aggiungere a un applet. Nei Capitoli 22 e 23 vengono fornite informazioni dettagliate sui diversi componenti dell'AWT.

Etichette

Il componente `Label` è una stringa visualizzata nel contenitore che non può essere modificata dall'utente. Nella Figura 7.3 è mostrato l'esempio di un applet con un'etichetta a fianco di un campo di testo.

Figura 7.3
Un'etichetta a sinistra di un campo di testo.

Per creare un componente `Label` e per aggiungerlo nella finestra di un applet si utilizza il seguente codice:

```
Label l = new Label("Indirizzo e-mail: ");
add(l);
```

Il parametro nel costruttore `Label("Indirizzo e-mail: ")` identifica il testo da visualizzare.

Campi di testo

Il componente `TextField` è una casella per l'input in cui un utente può digitare una singola riga di testo. È possibile configurare il numero di caratteri visibili nel campo di testo. Nella Figura 7.4 è mostrato l'esempio di un applet con un campo di testo.

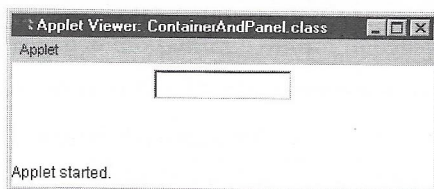
I componenti `TextField` vengono aggiunti nelle finestre nello stesso modo dei pulsanti o di qualsiasi altro componente:

```
TextField t = new TextField(12);
add(t);
```

Il parametro 12 nel costruttore `TextField(12)` imposta il campo di testo in modo che vengano visualizzati solamente 12 caratteri (circa). L'utente può digitare più caratteri, ma solo alcuni di essi vengono visualizzati.

Il numero effettivo di caratteri visualizzati dipende da quali caratteri vengono digitati. Un campo di testo può visualizzare più "i" che "m", in quanto i caratteri "i" sono più stretti dei caratteri "m".

Se come parametro viene specificata una stringa, come ad esempio `TextField t = TextField("nome")`, il campo di testo viene creato con quella stringa come testo predefinito nell'area di input del campo.

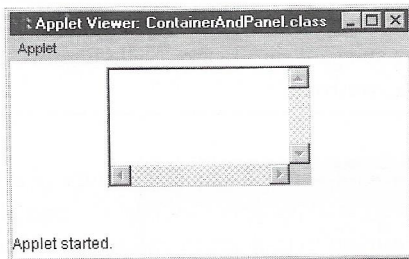
Figura 7.4
Un componente TextField.

È possibile specificare contemporaneamente sia il testo predefinito che la larghezza, utilizzando un'istruzione come la seguente:

```
TextField paese = new TextField("Stati Uniti", 20)
```

Per i campi di testo non esiste una lunghezza predefinita. Se non si definisce il parametro della lunghezza, si ottiene un campo di testo lungo un carattere.

Aree di testo

Il componente `TextArea` è una casella per l'input estesa che rende possibile l'inserimento di più righe di testo da parte dell'utente. È possibile configurare il numero di righe e di caratteri per riga visibili nell'area di testo. Nella Figura 7.5 è mostrato l'esempio di un applet con un'area di testo.

Figura 7.5 *Un componente TextArea.*

Per creare un componente `TextArea` e per aggiungerlo nella finestra di un applet, si utilizza il seguente codice:

```
TextArea t = new TextArea(5,20);
add(t);
```

I parametri `5` e `20` specificano rispettivamente il numero di righe e il numero di caratteri per riga dell'area di testo. Se il testo si estende oltre i bordi dell'area di testo, viene visualizzata una barra scorrevole che permette all'utente di scorrere le diverse sezioni dell'area.

Se come parametro del costruttore viene specificata una stringa, come nell'istruzione `TextArea t = TextArea("Era una notte buia e tempestosa.", 7, 25);`, l'area di testo viene creata con la stringa come testo predefinito nell'area di input del campo. Per fare in modo che determinate parti di questo testo inizino sulla riga successiva, si utilizza il carattere di nuova riga `\n` nel testo del parametro. Ad esempio, per inserire il testo "buia e tempestosa" all'inizio di una nuova riga, si utilizza la seguente istruzione:

```
TextArea t = TextArea("Era una notte \nbuia e tempestosa", 7, 25);
```

Pulsanti

Il componente `Button` è un pulsante rettangolare su cui è possibile fare clic con il mouse. L'applet `ColorCycle`, presentata precedentemente in questo capitolo, utilizzava un componente `Button` per modificare il colore dello sfondo.

Per creare un componente `Button` e per aggiungerlo nella finestra di un applet sono necessarie due righe di codice:

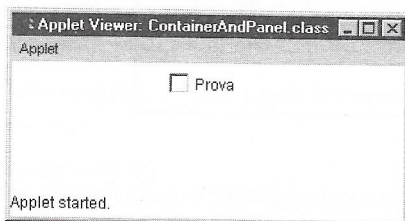
```
Button b = new Button("Annulla");
add(b);
```

Non facendo riferimento a un oggetto contenitore specifico, il metodo `add(b)` semplicemente aggiunge il pulsante alla superficie dell'applet. È anche possibile creare un nuovo pannello e aggiungervi il componente `Button`:

```
Panel p = new Panel();
Button b = new Button("Annulla");
p.add(b);
```

Caselle di controllo

Il componente `Checkbox` è una casella che può essere selezionata o deselezionata. Quando è selezionata, la casella di controllo visualizza un segno di spunta. Questo elemento di norma ha una riga di testo a fianco che ne spiega il significato. Nella Figura 7.6 è mostrato l'esempio di un applet con una casella di controllo.

Figura 7.6
Un componente Checkbox.

Per creare un componente `Checkbox` e per aggiungerlo nella finestra di un applet, si utilizza il seguente codice:

```
Checkbox c = new Checkbox("Prova");
add(c);
```

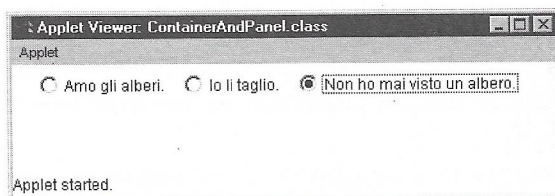
Il parametro nel costruttore `Checkbox("Prova")` identifica il testo da visualizzare. Se si omette questo parametro, la casella di controllo viene visualizzata senza alcun testo a fianco.

Le caselle di controllo vengono spesso utilizzate in un gruppo di caselle correlate, in cui è possibile selezionarne una sola alla volta. In HTML, si parla di pulsanti di opzione. *Per creare un gruppo di caselle di controllo si utilizza la classe* `CheckboxGroup`. *Il metodo* `setCheckboxGroup()` *associa una casella di controllo a un gruppo particolare. Il metodo* `setCurrent()` *di* `CheckboxGroup` *viene utilizzato per impostare la casella che deve risultare selezionata. Il Listato 7.5 mostra l'utilizzo di* `Checkbox` *e di* `CheckboxGroup`; *il codice si trova anche nel CD-ROM allegato al libro.*

Listato 7.5 *Un applet che crea un gruppo di caselle di controllo simile a un gruppo di pulsanti di opzione HTML.*

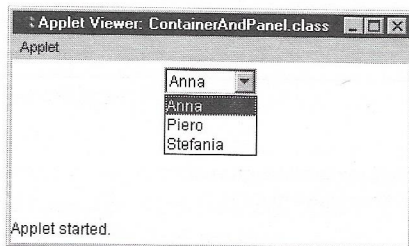
```
 1: import java.applet.Applet;
 2: import java.awt.*;
 3:
 4: public class ContainerAndPanel extends Applet {
 5:     CheckboxGroup cbg = new CheckboxGroup();
 6:
 7:     public void init() {
 8:         Checkbox c1 = new Checkbox("Amo gli alberi.");
 9:         c1.setCheckboxGroup(cbg);
10:         c1.setState(false);
11:         add(c1);
12:
13:         Checkbox c2 = new Checkbox("Io li taglio.",cbg,false);
14:         add(c2);
15:
16:         Checkbox c3 = new Checkbox("Non ho mai visto un albero.",cbg,true);
17:         add(c3);
18:     }
19: }
```

Si noti la differenza tra le righe da 8 a 10 e la riga 13. La riga 8 crea la casella di controllo con la relativa etichetta, la riga 9 la aggiunge al gruppo, la riga 10 ne imposta lo stato iniziale su `false`. La riga 13 combina queste tre operazioni in un'unica istruzione unendo i tre parametri nell'istanziazione del metodo di `Checkbox`. La Figura 7.7 mostra il gruppo di caselle di controllo creato con il Listato 7.5.

Figura 7.7 *Un gruppo di componenti Checkbox.*

In base alle impostazioni predefinite, quando vengono istanziate, tutte le caselle di controllo sono impostate su `false` (non selezionate). Non è necessario utilizzare `setState()` per le nuove caselle di controllo, a meno che si desideri verificare o impostare su `true` lo stato.

Menu di scelta

Il componente `Choice` è un elenco a discesa di stringhe in cui è possibile selezionare una sola stringa, come avviene per i gruppi di caselle di controllo. I menu di scelta forniscono un gruppo di opzioni e permettono di selezionarne solo una alla volta. La Figura 7.8 mostra l'esempio di un applet con un menu di scelta.

Figura 7.8
Un componente Choice.

Per creare un menu di scelta si utilizza il metodo `addItem()` della classe `Choice`. Per creare un elenco, lo si istanzia, vi si aggiungono i singoli elementi e quindi lo si aggiunge nella finestra dell'applet:

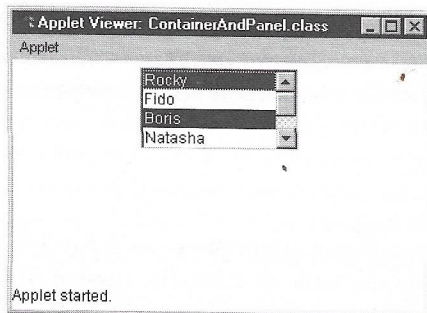
```
Choice c = new Choice();
c.addItem("Anna");
c.addItem("Piero");
c.addItem("Stefania");
add(c);
```

Elenchi a scorrimento

Il componente `List` è un elenco scorrevole in cui è possibile selezionare una o più stringhe. La Figura 7.9 mostra l'esempio di un applet con elenco a scorrimento.

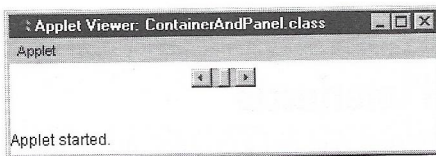
Per creare un elenco a scorrimento si utilizza il metodo `addItem()` della classe `List`. Per creare un elenco a scorrimento, per aggiungervi degli elementi e quindi per aggiungere il menu nella finestra di un applet, si utilizza il seguente codice:

```
List l = new List(4,true);
l.addItem("Rocky");
l.addItem("Fido");
l.addItem("Boris");
l.addItem("Natasha");
l.addItem("Du-Du-Du");
l.addItem("Lea");
add(l);
```

Figura 7.9
Un componente List.

Il costruttore `List()` può essere utilizzato senza parametri oppure con due parametri. L'istruzione `List l = new List(4, true)` utilizza due parametri: il primo indica il numero di elementi da visualizzare nella finestra dell'elenco, mentre il secondo determina quanti elementi possono essere selezionati. Se il secondo parametro è `true`, è possibile selezionare diversi elementi; diversamente è possibile effettuare una sola selezione alla volta, come per il menu di scelta.

Barre di scorrimento

Il componente `Scrollbar` è un quadratino che scorre dall'alto verso il basso o da sinistra a destra che può essere utilizzato per impostare un valore numerico. Questo componente viene utilizzato facendo clic con il mouse su una freccia o trascinando il quadratino scorrevole. La Figura 7.10 mostra l'esempio di un applet con una barra di scorrimento.

Figura 7.10 *Un componente Scrollbar.*

Il primo parametro del costruttore `Scrollbar` determina se si tratta di una barra di scorrimento verticale od orizzontale. Se si desidera impostare una barra verticale, si utilizza la costante `Scrollbar.VERTICAL`, diversamente si utilizza la costante `Scrollbar.HORIZONTAL`. Vi sono altri quattro parametri che determinano l'impostazione iniziale della barra scorrevole, le dimensioni dell'area di scorrimento, il valore minimo e il valore massimo.

Si consideri la seguente istruzione:

```
Scrollbar sb = new Scrollbar(Scrollbar.HORIZONTAL, 50, 500, 0, 1000);
add(sb);
```

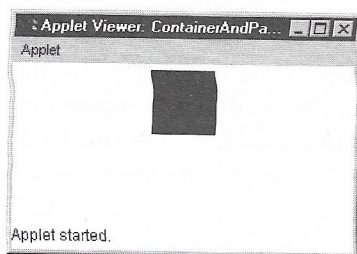
Questo codice crea un componente `Scrollbar` orizzontale con il quadratino inizialmente impostato su `50`. Quando l'utente muove il quadratino, l'area di scorrimento può restituire un valore compreso tra `0` (il valore minimo a sinistra) e `1000` (il valore massimo a destra). Il valore viene incrementato in passaggi di 2, in quanto le dimensioni totali della barra di scorrimento sono limitate a `500` elementi (metà del valore massimo).

Canvas

Il componente `Canvas` è una sezione di una finestra utilizzata principalmente come posto in cui disegnare la grafica o visualizzare immagini. Sotto questo punto di vista, un canvas assomiglia maggiormente a un contenitore che a un componente, tuttavia non può essere utilizzato per inserirvi dei componenti. Il seguente codice crea un componente `Canvas`, ne modifica le dimensioni a 50x50 pixel, ne imposta lo sfondo sul colore nero e lo aggiunge alla finestra di un applet:

```
Canvas c = new Canvas();
c.resize(50,50);
c.setBackground(Color.black);
add(c);
```

La Figura 7.11 mostra il risultato, così come appare all'utente: un'area nera sullo sfondo bianco di un applet.

Figura 7.11
Un componente Canvas.

Organizzazione dell'interfaccia

Finora, tutti i componenti sono stati aggiunti a un contenitore in modo simile a quello in cui vengono disposti gli elementi HTML in una pagina Web. Tutti gli oggetti sono organizzati semplicemente da sinistra a destra e dall'alto verso il basso e l'aspetto del contenitore dipende in grossa misura dalle dimensioni dell'area di visualizzazione.

Questo approccio è un modo *semplice per implementare l'indipendenza dalla piattaforma* di Java. Poiché il linguaggio deve funzionare in ogni sistema che ha un'implementazione di Java, l'ambiente a finestre deve essere flessibile. Ciò significa che uno stesso applet di Java sembra profondamente diverso quando viene visualizzato in un sistema Windows 95, piuttosto che in un sistema Macintosh o in una workstation SPARC, nonostante funzioni esattamente nello stesso modo.

Tuttavia, gli sviluppatori di Java hanno capito la necessità di organizzare i componenti di un'interfaccia utente e hanno ampliato le possibilità di Java di personalizzare l'interfaccia di un applet, in modo che questo funzioni su tutte le piattaforme e che abbia sempre un aspetto simile.

Gli strumenti aggiunti per questo scopo sono chiamati *gestori di layout*. Nei paragrafi precedenti, quando sono stati aggiunti i componenti in un contenitore (la finestra principale dell'applet), è stato utilizzato il layout predefinito: una classe chiamata `FlowLayout`. Esistono quattro altri gestori di layout che è possibile utilizzare per organizzare le interfacce: `BorderLayout`, `GridLayout`, `GridBagLayout` e `CardLayout`.

La classe FlowLayout

La classe `FlowLayout` è il gestore di layout predefinito per tutti i pannelli, inclusa la classe `Applet`, ed è il più semplice da utilizzare. I componenti posizionati secondo le regole del

gestore FlowLayout vengono disposti in ordine da sinistra a destra. Se un componente è troppo grande per poter essere inserito nella riga corrente, viene iniziata una nuova riga di componenti sotto la prima riga.

Ogni riga di componenti può essere allineata a sinistra, a destra o centrata. L'unico parametro utilizzato con il metodo add() è il nome dell'oggetto da aggiungere.

La seguente istruzione setLayout() imposta un contenitore in modo che utilizzi il gestore FlowLayout:

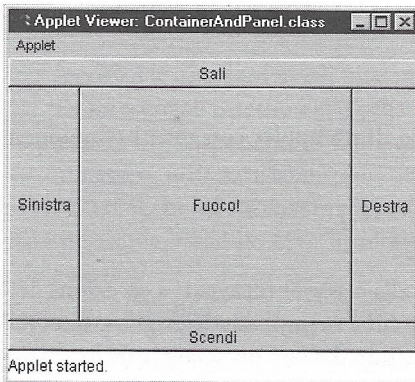
```
setLayout( new FlowLayout() );
```

La classe BorderLayout

La classe BorderLayout è il gestore di layout predefinito per tutte le classi Window, Dialog e Frame. In questo layout, i componenti vengono aggiunti ai bordi del contenitore, mentre tutto lo spazio rimanente viene raccolto nell'area centrale. Il metodo add() utilizza un parametro, una stringa che può essere North, South, East, West o Center, che specifica la posizione dei componenti.

Ad esempio, le istruzioni seguenti creano cinque pulsanti e li aggiungono in un contenitore organizzato con il gestore BorderLayout, come indicato nella Figura 7.12:

```
Button b1 = new Button("Sali");
Button b2 = new Button("Scendi");
Button b3 = new Button("Sinistra");
Button b4 = new Button("Destra");
Button b5 = new Button("Fuoco!");
setLayout( new BorderLayout() );
add("North", b1);
add("South", b2);
add("West", b3);
add("East", b4);
add("Center", b5);
```

Figura 7.12
Componenti disposti con BorderLayout.

Si noti l'utilizzo del metodo `setLayout()` per selezionare un gestore di layout per il contenitore. L'unico parametro utilizzato è un'istanza del layout desiderato. È importante ricordare di scrivere i parametri della direzione del metodo `add()` con l'iniziale maiuscola; come per molti altri aspetti di Java, per i quali è necessario fare attenzione alle lettere maiuscole e minuscole, per il gestore `BorderLayout` le direzioni devono sempre avere la lettera maiuscola (`North`, `South`, `East`, `West` e `Center`).

La classe GridLayout

La classe `GridLayout` inserisce ogni componente in una cella di una griglia a celle uniformi. Le dimensioni della griglia vengono definite quando la si crea e i componenti vi vengono aggiunti in ordine, iniziando dall'angolo superiore sinistro, in modo simile a come vengono aggiunti i componenti con il gestore `FlowLayout`, ma in questo caso a ogni componente viene dato lo stesso quantitativo di spazio nel contenitore.

I componenti aggiunti a un contenitore con il gestore `GridLayout` vengono disposti in ordine da sinistra a destra. Quando in una riga non vi sono più celle vuote, il componente successivo viene inserito nella cella di sinistra della riga successiva. A mano a mano che si rendono necessarie, vengono aggiunte nuove righe: se si crea una griglia con 3 righe e 3 colonne di celle e si aggiunge un decimo elemento, viene creata una quarta riga. L'unico parametro utilizzato con il metodo `add()` è il nome dell'oggetto da aggiungere.

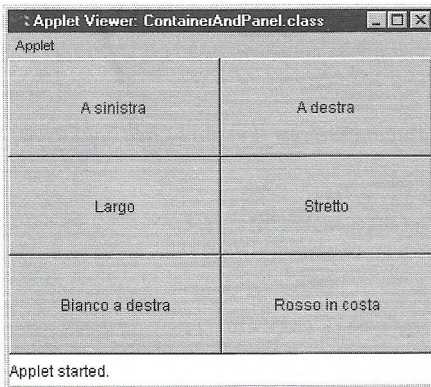
Le seguenti istruzioni `setLayout()` impostano un contenitore che utilizza il gestore `GridLayout` con tre righe e due colonne e quindi vi aggiunge una serie di pulsanti:

```
Button b1 = new Button("A sinistra");
Button b2 = new Button("A destra");
Button b3 = new Button("Largo");
Button b4 = new Button("Stretto");
Button b5 = new Button("Bianco a destra");
Button b6 = new Button("Rosso in costa");
setLayout( new GridLayout(3, 2));
add(b1);
add(b2);
add(b3);
add(b4);
add(b5);
add(b6);
```

La Figura 7.13 mostra l'esempio di un applet con tutti i componenti disposti secondo le regole del gestore `GridLayout`.

La classe GridBagLayout

La classe `GridBagLayout` è simile alla classe `GridLayout`, a eccezione del fatto che permette un controllo maggiore del modo in cui è organizzata la griglia e di come vengono presentati i componenti. Le celle nella griglia non devono occupare tutte lo stesso spazio e i componenti possono essere allineati in modo diverso in ogni cella.

Figura 7.13 *Componenti disposti con GridLayout.*

Per determinare come inserire un componente in una cella e quanto spazio deve occupare quest'ultima, si utilizza un oggetto speciale `GridBagConstraints`. A differenza dei gestori `FlowLayout` e `GridLayout`, il gestore `GridBagLayout` permette di aggiungere componenti nella griglia in qualsiasi ordine. Il primo passaggio per utilizzare il gestore `GridBagLayout` consiste nell'impostare il layout e l'oggetto `GridBagConstraints`, come indicato di seguito:

```
GridBagLayout gl = new GridBagLayout();
setLayout ( gl );
GridBagConstraints gb = new GridBagConstraints();
```

Prima di poter aggiungere un componente nel contenitore, si utilizzano variabili istanza dell'oggetto `GridBagConstraints` per determinare la posizione e l'allineamento del componente all'interno della cella. È possibile impostare le seguenti variabili di `GridBagConstraints`.

- `gridx` e `gridy`: queste variabili specificano la cella della griglia in cui si desidera inserire il componente; `gridx` rappresenta le righe e `gridy` rappresenta le colonne. La posizione `(1,1)` corrisponde all'angolo superiore sinistro.
- `gridheight` e `gridwidth`: queste variabili specificano il numero di celle che un componente dovrebbe occupare. `gridheight` determina il numero di righe e `gridwidth` determina il numero di colonne.
- `fill`: questa variabile specifica le direzioni in cui un componente deve espandersi se non ha spazio sufficiente per crescere all'interno della cella in cui è contenuto. Ciò può avvenire a causa della presenza di componenti di dimensioni maggiori in altre celle. Con questa variabile possono essere utilizzate le costanti `GridBagConstraints.HORIZONTAL`, `GridBagConstraints.VERTICAL`, `GridBagConstraints.BOTH` e `GridBagConstraints.NONE`. Il seguente esempio imposta la variabile `fill` in modo che si espanda in entrambe le direzioni orizzontale e verticale:

  ```
  gb.fill = GridBagConstraints.BOTH;
  ```

- `anchor`: questa variabile specifica il modo in cui un componente deve essere allineato all'interno della cella che lo contiene. Vengono utilizzate le seguenti costanti di `GridBagConstraints`: `NORTH`, `NORTHEAST`, `EAST`, `SOUTHEAST`, `SOUTH`, `SOUTHWEST`, `WEST`, `NORTHWEST` e `CENTER`.

Il Listato 7.6 mostra l'esempio di un componente inserito in una pagina utilizzando la classe `GridBagLayout`*. Il codice si trova anche nel CD-ROM allegato al libro.*

Listato 7.6 *Inserimento di un componente con il gestore GridBagLayout.*

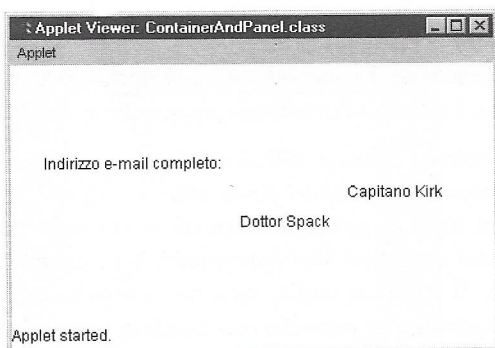
```
// dichiara variabili
GridBagLayout gl = new GridBagLayout();
GridBagConstraints gb = new GridBagConstraints();
// sceglie un gestore di layout
setLayout ( gl );
// crea componenti
Label l1 = new Label("Indirizzo e-mail completo:");
Label l2 = new Label("Capitano Kirk");
Label l3 = new Label("Dottor Spack");
// imposta i vincoli e aggiunge componenti
gb.gridx = 1;
gb.gridy = 1;
gb.gridwidth = 2;
gl.setConstraints(l1, gb);
add(l1);
// imposta i vincoli e aggiunge componenti
gb.gridx = 5;
gb.gridy = 3;
gb.gridwidth = 2;
gl.setConstraints(l2, gb);
add(l2);
// imposta i vincoli e aggiunge componenti
gb.gridx = 3;
gb.gridy = 5;
gb.gridwidth = 2;
gl.setConstraints(l3, gb);
add(l3);
```

In questo esempio, il primo componente viene inserito nella riga 1 e colonna 1 della griglia e occupa due celle in larghezza; il secondo componente viene inserito nella riga 5 e colonna 3; il terzo nella riga 3 e colonna 5. La Figura 7.14 mostra l'esempio di un applet in cui tutti i componenti sono disposti secondo le regole del gestore `GridBagLayout` e dell'oggetto `GridBagConstraints`.

La classe CardLayout

La classe `CardLayout` è uno speciale gestore di layout. Anziché visualizzare simultaneamente diversi pannelli, crea una pila di pannelli che possono essere visualizzati uno alla volta, come avviene con il mazzo di carte nel gioco del Solitario. La classe `CardLayout` ha un proprio gruppo di metodi che vengono utilizzati per controllare quale pannello deve essere visualizzato.

Il Listato 7.7 utilizza una serie di pannelli con canvas di diversi colori per mostrare la classe `CardLayout`*. Facendo clic nell'area dell'applet si fa in modo che il gestore di layout carichi il pannello successivo. Se è già stato caricato l'ultimo pannello, il gestore* `CardLayout` *torna automaticamente al primo. Questo codice si trova anche nel CD-ROM allegato al libro.*

Figura 7.14 *Componenti disposti con GridBagLayout.*

Listato 7.7 *Utilizzo della classe CardLayout per visualizzare tre pannelli di colore diverso.*

```
import java.applet.Applet;
import java.awt.*;

public class cardStack extends Applet {
     private Panel   canvasCards = new Panel(),
        p1 = new Panel(), p2 = new Panel(), p3 = new Panel();
     private CardLayout cardDeck = new CardLayout();

    public void init() {
        canvasCards.setLayout( cardDeck );

        p1.setLayout (new BorderLayout());
        p2.setLayout (new BorderLayout());
        p3.setLayout (new BorderLayout());

        Canvas c1 = new Canvas(), c2 = new Canvas(), c3 = new Canvas();
        c1.setBackground(Color.black);
        c2.setBackground(Color.red);
        c3.setBackground(Color.green);

        p1.add("Center", c1);
        p2.add("Center", c2);
        p3.add("Center", c3);

        canvasCards.add("p1", p1);
        canvasCards.add("p2", p2);
        canvasCards.add("p3", p3);

        setLayout(new BorderLayout());
        add("Center", canvasCards);
    }

    public boolean mouseDown(Event event, int x, int y) {
        cardDeck.next(canvasCards);
        return true;
    }
}
```

Associazione di layout con pannelli annidati

Nonostante l'Abstract Windowing Toolkit offra diversi tipi di gestori di layout, spesso un unico gestore specifico non è adatto all'interfaccia utente che si desidera creare.

La soluzione a questo problema consiste nell'annidare un tipo di contenitore dentro un altro, in modo che il contenitore annidato possa avere un tipo di gestore di layout e il contenitore più grande un altro. È possibile annidare tutti i contenitori che si desidera, perché i contenitori possono contenere altri contenitori; ogni contenitore può utilizzare il proprio gestore di layout. Il seguente codice crea un contenitore che utilizza il gestore `BorderLayout` e quindi vi aggiunge un pannello con caselle di controllo. Poiché per le caselle non è specificato nessun gestore di layout, viene utilizzato il gestore predefinito `FlowLayout`.

```
setLayout( new BorderLayout() );
Button b1 = new Button("Acquista");
add("North", b1);
Button b2 = new Button("Esci");
add("West", b2);
Button b3 = new Button("Guida");
add("East", b3);
Button b4 = new Button("Sfoglia catalogo");
add("South", b4);
Panel p = new Panel();
// add check boxes to panel
Checkbox c1 = new Checkbox("Piatto: L. 10.000");
p.add(c1);
Checkbox c2 = new Checkbox("Pentola: L. 23.000");
p.add(c2);
Checkbox c3 = new Checkbox("Coltello: L. 15.000");
p.add(c3);
// aggiunge il pannello al contenitore principale
add("Center",p);
```

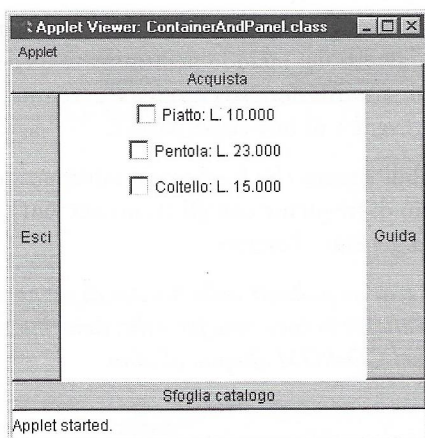
La Figura 7.15 mostra un applet che utilizza questo codice. Utilizzando i pannelli annidati e diversi tipi di gestori di layout, è possibile creare qualsiasi tipo di finestre per l'interfaccia utente.

Gestione degli eventi

A questo punto si hanno gli strumenti necessari per creare interfacce utente di effetto per gli applet di Java. È possibile utilizzare campi di testo per immettere caratteri, fare clic su pulsanti, spostare barre di scorrimento e così via.

Tuttavia, l'interfaccia utente non ha modo di rispondere a nessuno di questi input dell'utente. In un ambiente a finestre come quello fornito dall'Abstract Windowing Toolkit, l'input dell'utente tramite il mouse, la tastiera o altri strumenti, genera un evento.

Un *evento* è un modo in cui un programma comunica che è avvenuto qualcosa. Gli eventi generano automaticamente chiamate a metodi nello stesso modo in cui può essere richiamato automaticamente un metodo `paint()` quando deve essere aggiornata la finestra di un applet o un metodo `init()` quando viene eseguito per la prima volta un applet.

Figura 7.15
Un contenitore gestito con BorderLayout con un pannello annidato al centro gestito da FlowLayout.

Gli eventi possono coinvolgere elementi dell'interfaccia utente, ad esempio un pulsante su cui è stato fatto clic con il mouse, ma possono anche non essere correlati all'interfaccia, ad esempio nel caso di una condizione di errore che causa l'interruzione dell'esecuzione.

In Java, tutti gli eventi correlati all'interfaccia utente sono gestiti dalla classe `java.awt.Event`.

Al fine di controllare i componenti dell'interfaccia utente che possono essere creati per l'utilizzo con gli applet, vi sono due tipi di eventi da prendere in considerazione: di azione e di scorrimento.

Eventi di azione

Gli *eventi di azione* vengono generati dalla maggior parte dei componenti dell'interfaccia utente per segnalare che è avvenuto qualcosa. A seconda del componente, ciò può avere diversi significati.

- Per i pulsanti, un evento significa che è stato fatto clic sul pulsante.
- Per le caselle di controllo, un evento significa che la casella è stata selezionata o deselezionata.
- Per gli elenchi o i menu di scelta, un evento significa che uno degli elementi dell'elenco è stato selezionato.
- Per i campi di testo, un evento significa che è stato premuto il tasto Invio per indicare che l'utente ha completato l'inserimento.

Il metodo `action()` fa parte della classe `Event` e ha la seguente sintassi:

```
public boolean action(Event e, Object o) {
    // qui va il codice del metodo
}
```

Tutti i componenti dell'interfaccia utente che devono generare un evento di azione utilizzano il metodo `action()`. Due parametri del metodo determinano quale componente ha generato l'evento e raccolgono altre informazioni su che cosa è avvenuto. Questi due parametri sono un'istanza di una classe `Event` e di una classe `Object`.

La classe `Event` ha diverse variabili istanza che forniscono informazioni sull'evento che ha avuto luogo. Quella utilizzata più di frequente con gli eventi `action()` è la variabile `target`, che indica quale componente ha generato l'evento.

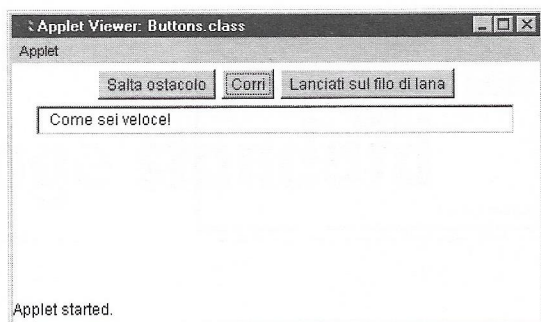
Il codice del Listato 7.8 crea tre pulsanti nella finestra di un applet e imposta un campo di testo in risposta al pulsante su cui è stato fatto clic, come indicato nella Figura 7.16. Il codice si trova anche nel CD-ROM allegato al libro.

Listato 7.8 *Il codice sorgente completo di Buttons.java.*

```
 1: import java.awt.*;
 2:
 3: public class Buttons extends java.applet.Applet {
 4:     Button b1 = new Button("Salta ostacolo");
 5:     Button b2 = new Button("Corri");
 6:     Button b3 = new Button("Lanciati sul filo di lana");
 7:     TextField t = new TextField(50);
 8:
 9:     public void init() {
10:         add(b1);
11:         add(b2);
12:         add(b3);
13:         add(t);
14:     }
15:
16:     public boolean action(Event e, Object o) {
17:         if (e.target instanceof Button) {
18:             String s = (String)o;
19:             if ( s.equals("Salta ostacolo") )
20:                 t.setText("Un bel salto, complimenti.");
21:             else if ( s.equals("Corri") )
22:                 t.setText("Gran bella velocità!");
23:             else
24:                 t.setText("Hai vinto? Chissà!");
25:             return true;
26:         }
27:         return false;
27:     }
28: }
```

Eventi di scorrimento

Il metodo `action()` viene generato da tutti i componenti descritti in questo capitolo, a eccezione di `Scrollbar`; questo componente utilizza il metodo `handleEvent()` che, a differenza del metodo `action()`, ha come unico parametro un'istanza della classe `Event`:

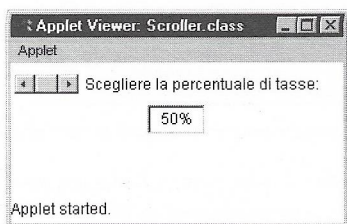
Figura 7.16
L'applet Buttons.

```
public boolean handleEvent(Event e) {
    // qui va il codice del metodo
}
```

Utilizzando la variabile `target` *della classe* `Event`*, è possibile determinare quale componente ha generato l'evento e rispondergli. Il Listato 7.9 imposta un valore in un campo di testo sulla base dell'input dell'utente tramite una barra di scorrimento. Il codice si trova nel CD-ROM allegato al libro.*

Listato 7.9 *Il codice sorgente completo di Scroller.java.*

```
 1: import java.awt.*;
 2:
 3: public class Scroller extends java.applet.Applet {
 4:     Scrollbar s = new Scrollbar(Scrollbar.HORIZONTAL,
 5:                 50,100,0,100);
 6:     Label l = new Label("Scegliere la percentuale di tasse: ");
 7:     TextField t = new TextField("50%", 3);
 8:
 9:     public void init() {
10:         add(s);
11:         add(l);
12:         add(t);
13:     }
14:
15:     public boolean handleEvent(Event e) {
16:         if (e.target instanceof Scrollbar) {
17:             int taxrate = ((Scrollbar)e.target).getValue();
18:             t.setText(taxrate + "%");
19:             return true;
20:         }
21:     return false;
22:     }
23: }
```

Quando si utilizza un browser Web o l'appletviewer del JDK per visualizzare l'applet completo, questo dovrebbe assomigliare alla Figura 7.17.

Figura 7.17
L'applet Scroller.

Riepilogo

L'API di Java include funzionalità che gestiscono automaticamente buona parte del lavoro. L'interfaccia utente fornisce molti componenti quali pulsanti, campi di testo e menu di scelta che è possibile estendere creando sottoclassi, senza che sia necessario scrivere molto codice.

I compiti che in alcuni linguaggi possono essere difficili, ad esempio l'animazione e la gestione degli eventi, in Java sono relativamente facili. Uno degli obiettivi originari del progetto di Java era la semplicità. Si potrebbe obiettare che la programmazione orientata agli oggetti non è mai facile, tuttavia la programmazione di applet è un buon punto di partenza per i principianti, in quanto è semplice sviluppare programmi utili per il Web senza scrivere molto codice.

Questa introduzione alla programmazione degli applet e all'Abstract Windowing Toolkit ha mostrato molte delle funzionalità di Java incluse nelle librerie di classi. Per molti applet, la creazione dell'interfaccia utente rappresenta il grosso del lavoro, in quanto buona parte del codice necessario per controllare l'interfaccia è già stato scritto.

Ogni componente dell'interfaccia utente ha metodi che possono essere utilizzati per richiamarne o modificarne i valori, per attivarne o disattivarne il funzionamento e per eseguire altri compiti. Nell'Appendice C viene fornito un elenco completo di questi metodi.